TLB misses - The Missing Issue of Adaptive Radix Tree?

Petrie Wong Ziqiang Feng Wenjian Xu Eric Lo Ben Kao
Department of Computer Science, The University of Hong Kong
Department of Computing, The Hong Kong Polytechnic University

Motivation

- In-memory databases
  - H-Store
  - Hekaton

- Efficient in-memory index structures
  - Cache-Sensitive B+-Tree (CSB+-Tree)
  - Fast Architecture Sensitive Tree (FAST)
  - Adaptive Radix Tree (ART)

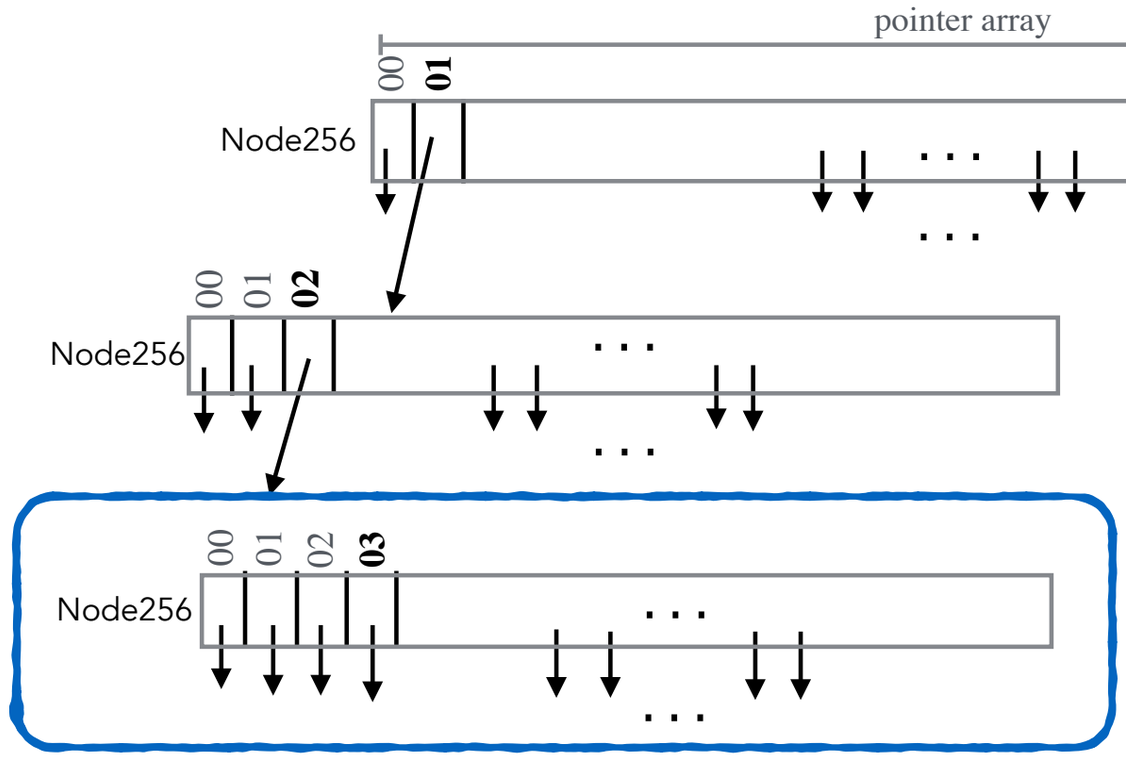
Why Adaptive Radix Tree

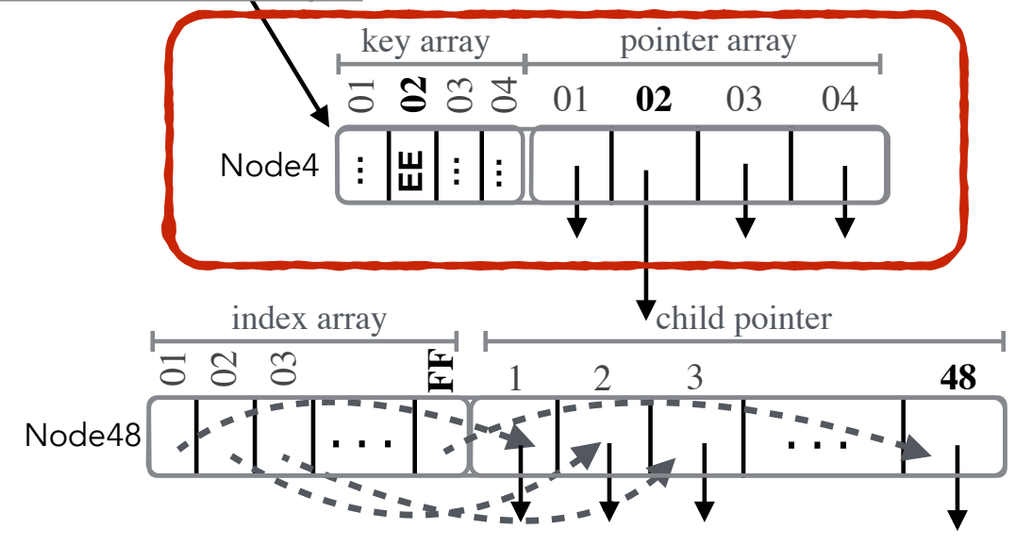
- Outperforms existing index structures
  - both search and update
  - has small memory footprint
  - Avoid cache miss
  - Leverage SIMD data parallelism
  - Reduce branch mis-prediction
    - adopt a radix tree structure

V. Leis, A. Kemper al et ICDE'13

What is Adaptive Radix Tree

small node type (Node4)
for nodes with few child pointers





large node type (Node256)
for nodes with many child pointers

Whether TLB miss matter in ART?

- Translation Look-aside Buffer (TLB)
  - cache for page table entries
  - fast way to translate virtual memory address to physical memory address
    - executing an instruction in CPU

for program

for CPU

- in-memory index structure like ART
  - few cache miss, few branch mis-prediction, SIMD-friendly
  - whether misses in TLB would become a bottleneck
    - if positive
      - what are the measures to alleviate
      - how effective those measures are

Whether TLB miss matter in ART?

- Experiment to show
  - Stall time % due to TLB miss
- System specification
  - Intel Core i7 2630QM CPU
  - 2.00 GHz clock rate, 2.9 GHz turbo frequency.
  - Each core
    - 32KB L1i cache, 32KB L1d cache, 256KB unified L2 Cache
  - share 6MB L3 cache, 16GB 1600 RAM.

Whether TLB miss matter in ART?

- Data
  - 1,000,000 integer keys
  - Dense: from 1 to n (19MB in RAM)
  - Sparse: random number in 32bit domain (22MB in RAM)
    - cannot fit into 6MB L3 cache

Whether TLB miss matter in ART?

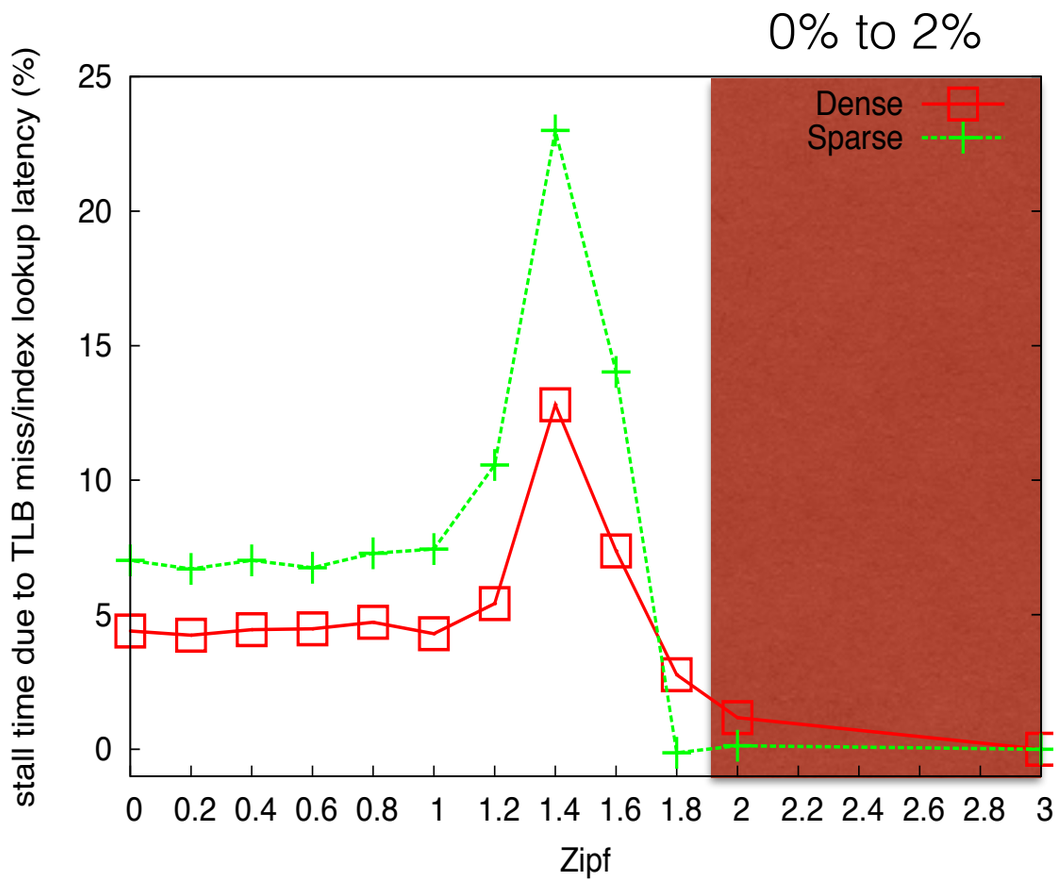
- Workload
  - 256M lookups
  - Varying skewness:
    - zipf=0 (each key is uniformly accessed)

      to

    - zipf=3 (few very hot keys and many non-hot keys)

Whether TLB miss matter in ART?

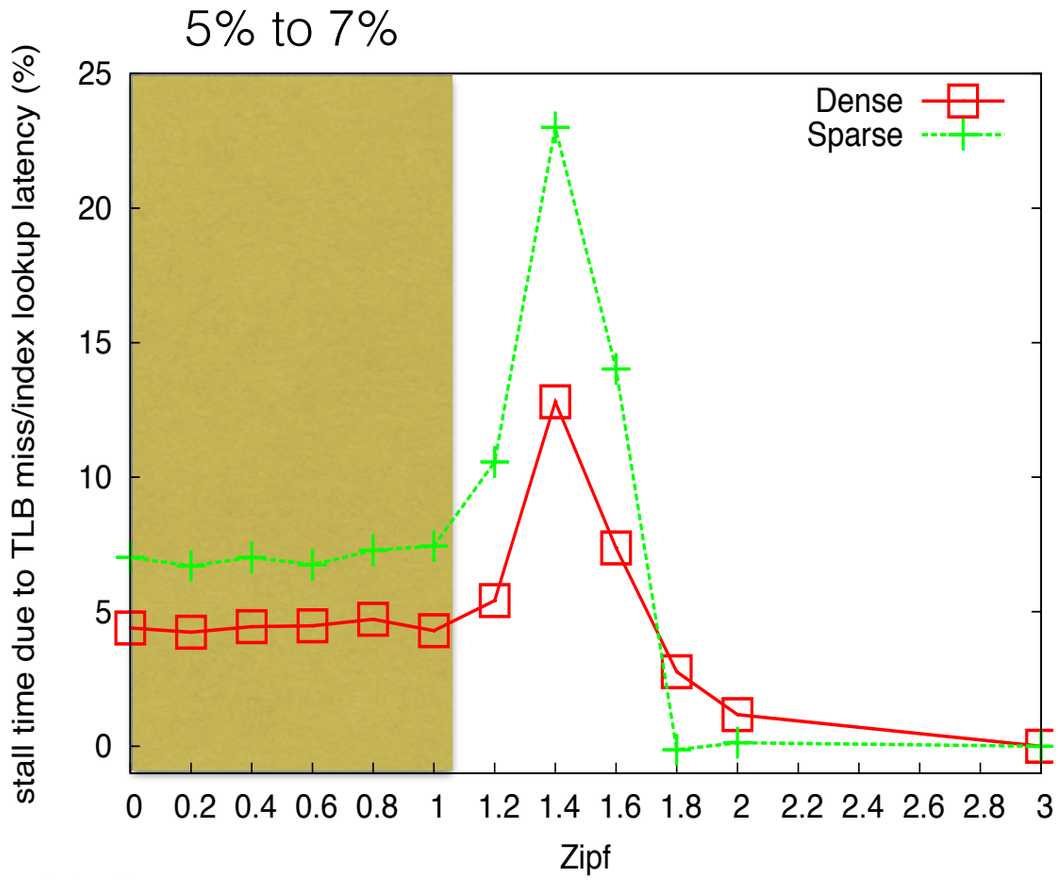
- No, when key access is very skew (Zipf=2 to 3)
  - few very hot search keys
    - occupies very few page table entries in TLB
      - very few TLB misses are incurred (0% to 2% of stall time)
      - TLB miss doesn't matter



Very skew

Whether TLB miss matter in ART?

- No, when workload is not skew (Zipf=0 to 1)
- each key is uniformly accessed
  - no spatial locality
  - lots of cache misses
    - dominate the latency
      - TLB miss not so matters (5% to 7%)



Uniform

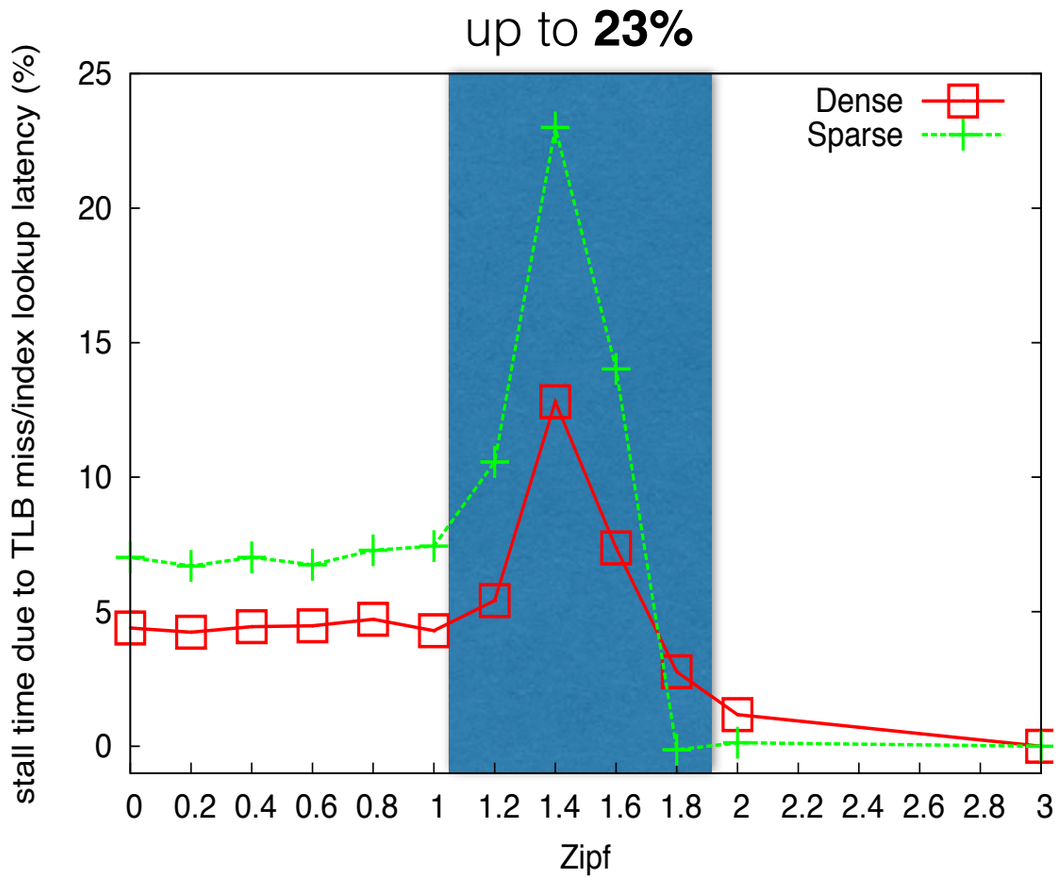
Whether TLB miss matter in ART?

- YES, when the workload posses realistic skewness (Zipf = 1 to 2)

- key access with certain spatial locality

  - cache miss is not high

  - TLB matters now (up to 23%)



What are the measures that we can take to alleviate?

- use of huge page
- workload-conscious node-to-page reorganization

What are the measures that we can take to alleviate?

- **use of huge page**

- workload-conscious node-to-page reorganization

What is Huge Page?

- In memory allocation
  - eliminate fragmentation over the whole memory space
    - cutting memory space into pages
      - Regular page size (in most processors e.g. Intel Sandy Bridge - Xeon E5)
        - 4KB
        - OS's default value
      - Huge page size (e.g. Sandy Bridge)
        - 2MB, 1GB
        - good tactic to reduce TLB misses

Why Huge Page?

- if apply huge page in ART
  - **reduce # of pages** spanned by ART nodes
    - reduce the pressure on the TLB
      - fewer TLB miss
        - throughput increase

Why Huge Page?

- page table entry
  - besides being stored in TLB
    - **occupy space in L1/L2/L3 cache** and RAM
- So… fewer page table entries
  - occupy fewer space in processor's cache
    - fewer cache misses
      - throughput increase

L2 Cache when using **regular** page

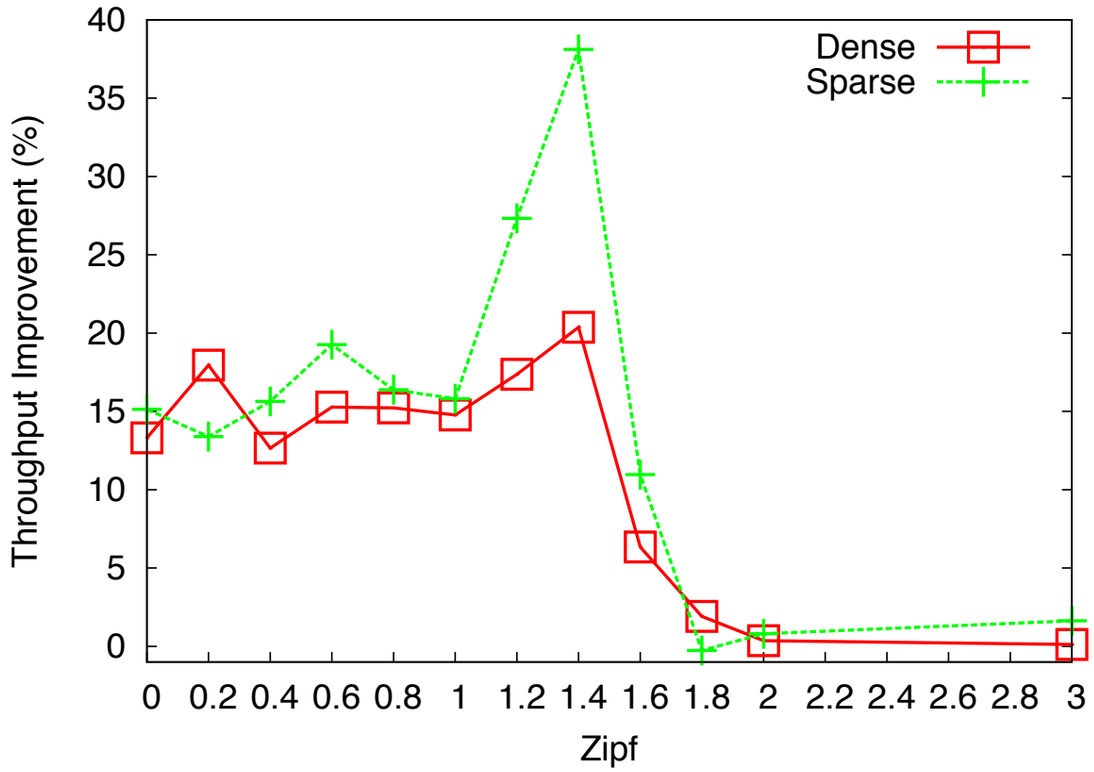
L2 Cache when using **huge** page

Huge Page always Help?

- but…
- different # of TLB entries for different page sizes
- # of **huge page entries** are **fewer** than that of regular page entries
- In Xeon E5,
  - 64 DTLB and 512 STLB entires for regular pages
  - 32 DTLB entires for huge pages
- fewer TLB entries available for huge page
  - throughput may decrease

Can Huge Page Help?

- Yes
- when workload is uniform and quite skew (Zipf < 2)
  - reduce TLB miss and cache miss
    - throughput increase as expected
- when workload extreme skew (Zipf > 2)
  - very few TLB miss and cache miss
    - no further improvement



What are the measures that we can take to alleviate?

- use of huge page

- **workload-conscious node-to-page reorganization**

What is Workload-Conscious Node-to-Page Reorganization?

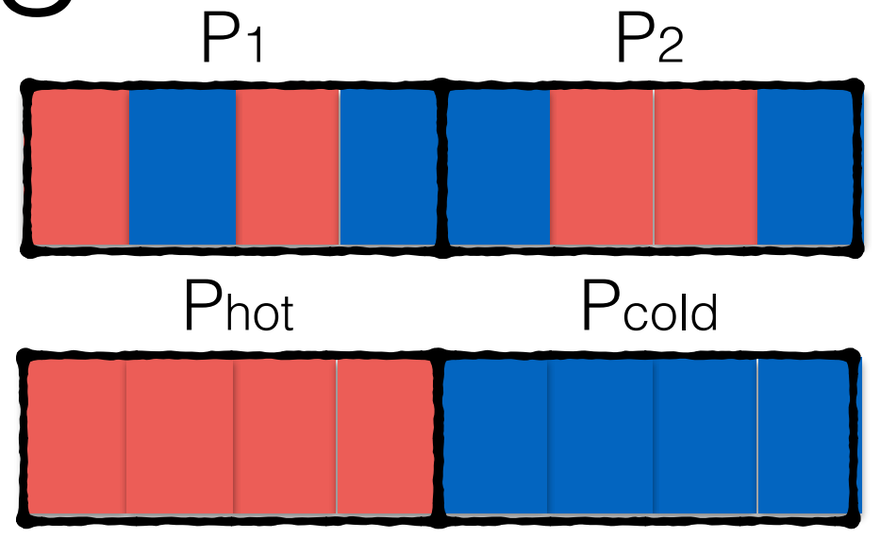
- tree nodes in ART are allocated
  - dynamic memory allocation
    - OS's default scheme
    - eliminate fragmentation over the whole memory space
  - workload-conscious allocation (R. Stoica and A. Ailamaki et al. DaMoN'13)
    - takes over OS's control
    - organize the hot ART nodes into the same page.

Why Workload-Conscious Node-to-Page Reorganization

- OLTP workload is skew
  - some keys are **hot** and accessed frequently
    - if putting all hot nodes into one (huge) page
      - page table entry of the hot page will be kept in TLB
        - no TLB miss when accessing hot keys
          - Throughput increase

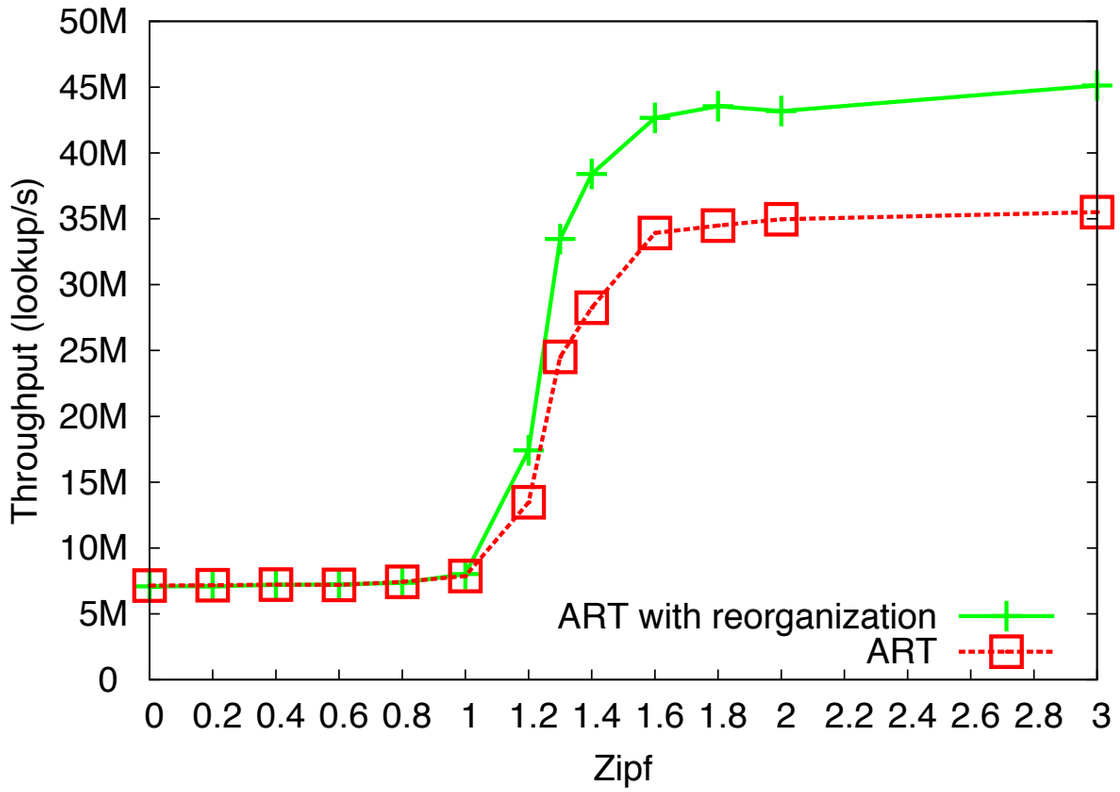
How Workload-Conscious Node-to-Page Reorganization

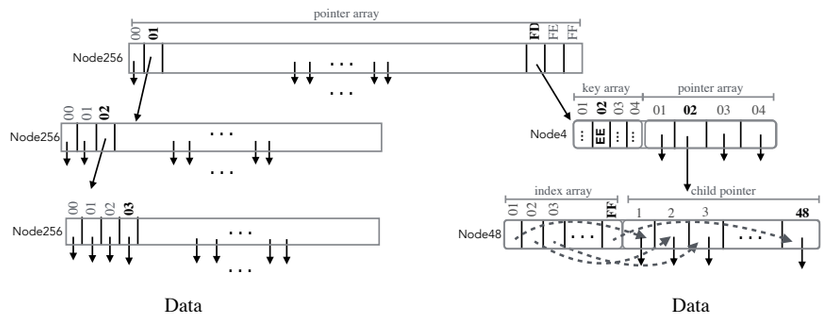
- When query execution
  - log key accesses
- analyzing access logs
  - sort the keys by their access frequencies
- node-to-page reorganization
  - according to access frequencies
  - hot nodes will be placed in same page



Can Workload-Conscious Node-to-Page Reorganization Help?

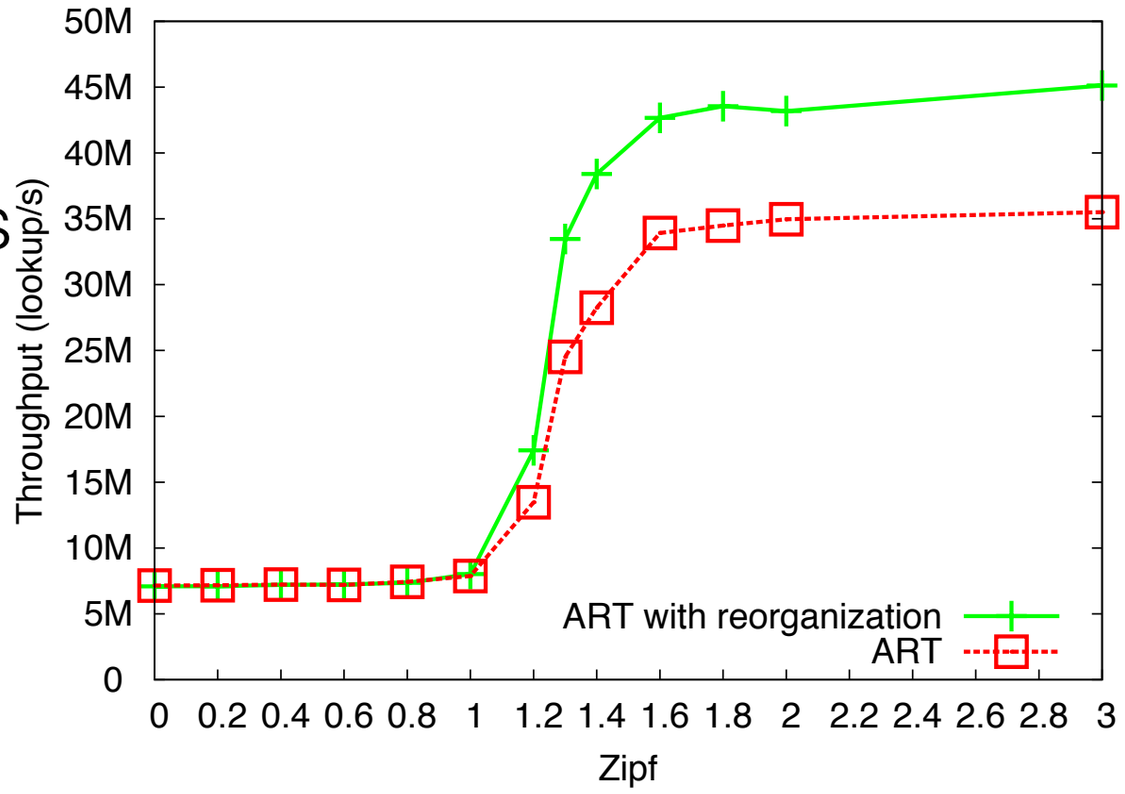
- Yes, when
  - data is sparse and workload is skew
- sparse data
  - each node contain few children
    - small nodes (Node4, size is 36 byte) are used
      - many nodes, not so condense
        - more space, more pages
        - more page table entries
          - TLB miss matters





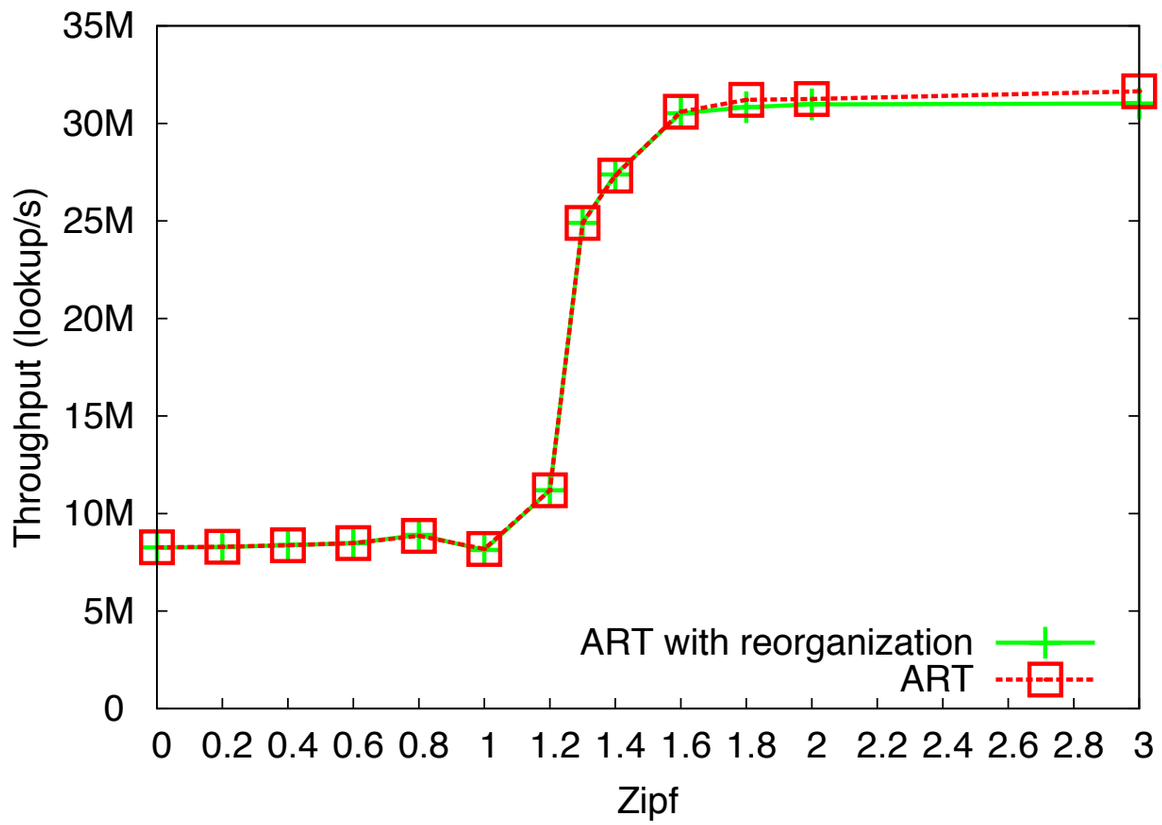
Can Workload-Conscious Node-to-Page Reorganization Help?

- sparse data
  - when workload-conscious reorganization applied
    - all hot nodes can put into few pages
      - fewer page table entries need to be cached (for hot nodes)
        - fewer TLB miss and throughput increase



Can Workload-Conscious Node-to-Page Reorganization Help?

- No, when
  - data is dense
  - huge page is used
  - be few pages needed.
    - all page table entries can stay in TLB
    - giving almost no TLB miss
      - make node-to-page reorganization immaterial



Summary

- TLB miss does matter when the access workload possess realistic skew

- the use of huge page provides 1-32% positive lookup throughput improvement over the use of regular page

- workload-conscious node-to-page reorganization does help when the data to be indexed is sparse

Thank you